Efficient Symbolic Task Planning for Multiple Mobile Robots

Yuqian Jiang

December 13, 2016

Abstract

Symbolic task planning enables a robot to make high-level decisions toward a complex goal by computing a sequence of actions with minimum expected costs. This thesis builds on a single-robot planning framework, and aims to address two issues: (1) lack of performance information in the selection of planners across different formalisms, and (2) time complexity of optimal planning for multiple mobile robots. In this thesis we first investigate the performance of the state-of-the-art solvers of Planning Domain Definition Language (PDDL) and Answer Set Programming (ASP) in robot navigation problems. We then aim to reduce overall costs where multiple mobile robots may block in narrow corridors or collaborate to open doors. It is challenging to model such interactions due to uncertain delays of navigation actions in populated areas. This paper addresses this challenge with an algorithm which calculates the conditional distribution of plan costs for each robot, given planned actions of other robots. We then propose an iterative conditional planning algorithm to efficiently approximate optimal plans of the system. Experiments in simulation and a demonstration on real robots show that the algorithm has a significant advantage over baselines in which robots plan individually, or plan together without a model for navigation action durations.

1 Introduction

As mobile robots become capable of autonomous navigation for extended periods of time in large spaces, it is often inefficient to plan the complete trajectory in one setting. Some robots are also designed to perform heterogeneous tasks, such as picking up objects and interacting with humans along the way. Symbolic planning, given abstracted descriptions of the environment, allows a robot to break down a complex task into a sequence of symbolic actions. Planning problems are often represented in action languages, which formalize the state transition system as action preconditions and effects. Many symbolic planners solve such problems by heuristic-based search of the state space specified by the action language encoding. Another approach is to convert action programs to Answer Set Programming (ASP) programs, and compute answer sets with techniques in satisfiability testing [12] [24]. Both approaches are able to consider action costs and compute the optimal plan, so the robot can reach the goal while minimizing the use of resources. In a dynamic environment where the state can change without involvement of the robot, for instance, where doors open and close by pedestrians, it is difficult to predict the exact state. At planning time, the robot may use defaults, and replan if further sensing negates the assumptions of the plan.

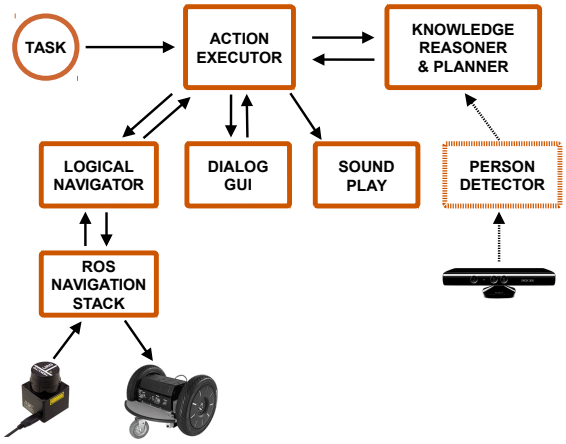
When multiple robots are operating in the same space, the action of one robot may affect the others, so their individual optimal plans can become overall suboptimal. For example, if two robots decide to take a narrow corridor in opposite directions, they will block each other in the corridor, and both robots will have to replan. Unlike the state of a door which is controlled unpredictable forces, interactions between robots can be predicted based on their planned actions. One conceivable solution is to encode all robots into the planning domain, along with action effects they may have on each other. In the example of a narrow corridor, if we explicitly state that navigating through the corridor has the effect of blocking it in the opposite direction, a symbolic planner can search for collision-free plans for all robots. There are, however, two problems with this approach. First, it is not scalable to the number of robots. Second, the corridor is blocked only when a robot is inside, and the entry and exit time may vary in practice. Since the robot has to move around humans and other movable objects in a populated domain, navigation actions can be delayed by an uncertain amount of time. Therefore, the goal of this research is to build an efficient multi-robot task planning system that scales well with the number of robots while addressing uncertainties in navigation time.

The rest of this thesis is organized as follows: Section 2 introduces the background of symbolic task planning, the development of action languages and planners, and their counterparts for multi-robot planning. The first part of Section 3 describes the existing hardware and software platform of the Building Wide Intelligence (BWI) robots, a group of autonomous robots designed to interact with people in the Computer Science building of UT Austin. We then introduce the representation of an example office domain in action languages, and select a fast planner based on an empirical comparison of the performance of state-of-the-art solvers. Section 4 focuses on solving the problem of symbolic planning for multiple mobile robots with our iterative conditional planning algorithms. Finally, we evaluate several configurations of the algorithm in simulations of the office domain, and demonstrate successful implementation on real robots.

2 Background

The history of using action languages to describe and solve automated planning problems dates back to the development of STRIPS [9] [13]. One of the most popular action languages today is the Planning Domain Definition Language (PDDL) [26]. PDDL is developed as a standard formalism to compare planner performances in International Planning Competition (IPC) [35]. Since then, many planning systems have been developed with state-of-the-art heuristics, such as Fast-Forward [17] and Fast-Downward [16]. On another front, efforts have been made to improve the representation capability of action languages and allow more general knowledge reasoning. For example, language BC supports the representation of defaults and non-monotonic reasoning [23]. Instead of explicitly searching the state space, action language solving can translate to Answer Set Programming [24], a language with wide applications in knowledge representation and reasoning, and fast solvers powered by satisfiability checking algorithms. Task planning problems for mobile robots can be described in action languages, such as PDDL and BC, and then solved by general purpose solvers for the specific language [30] [19].

(a) Three BWI robots.



(b) System architecture of the BWI robot.

3 Single-robot Planning System

This section explains how the BWI robots plan individually to accomplish navigation tasks. The system is the outcome of ongoing efforts in the BWI lab to provide a reliable research platform. The ASP formalization presented in Section 3.2.1 is adapted from previous work by Piyush Khandelwal, Fangkai Yang and Matteo Leonetti [39]. The empirical comparison of planners is joint work with Shiqi Zhang, Piyush Khandelwal, in addition to the thesis advisor, Peter Stone.

3.1 System Architecture

As shown in Figure 1a, the hardware architecture is built upon Segway Robotic Mobility Platform. Their sensors include Hokuyo URG-04LX laser range-finders, and Kinect RGB-D cameras [14]. Figure 1b shows the software system of the BWI robots which uses the Robot Operating System (ROS) [29]. The system integrates a symbolic task planner to make high-level decisions. When a task is assigned to a robot, the action executor first formalizes the goal, and sends a planning request to the planner. The planner computes a sequence of actions based on action rules and current knowledge of the domain. The action executor then carries out each action by invoking the corresponding modules in Figure 1b according to the action type. Navigation actions are handled by the logical navigator. It translates symbolic locations to physical coordinates, and sends the coordinates

to the navigation stack which ultimately gives velocity commands to the base controller. The logical navigator is also responsible for constructing the symbolic representation of sensor data, and returning new observations when each navigation action finishes. Other types of actions may involve different modules. For example, if the action is to ask if a person is in the current room, the robot may speak through the ROS `sound_play` node, and wait for user input from a dialog interface. Furthermore, observations do not necessarily come from navigation actions. Various sensor data can be parsed as input to the reasoner. Hypothetically, if we have a person detector running at all times, it may update locations of people to the knowledge base.

3.2 Choice of Planner

The purpose of this section is to investigate how planners perform in a robot navigation domain while varying features of planning requests, specifically the number of objects and the length of the optimal plan. Unlike classical planning domains such as Blocks World, realistic planning problems of robots often involve reasoning about many objects as well as state variables, and indirect or recursive action effects. We capture these features in a test domain derived from the environment of the BWI robots, and compare the performance of two state-of-the-art ASP and PDDL solvers.

3.2.1 Domain Description

The domain resembles an office environment which consists of rooms that are common areas or cubicles, and offices that are connected to each other via doors. Figure 2 shows an example floor plan of the described environment. A mobile robot can autonomously navigate between locations unless a closed door is on the path. Other navigation actions include approaching a door, and opening a door. The exact action the robot performs to open a door is domain dependent, either by using a robotic arm or asking for help.

The domain knowledge is formalized with predicates in ASP and PDDL. At a particular timestep of a plan, the value of every predicate together describes the full state of the domain. To make a fair comparison, we use the same set of predicates for both languages. Time-dependent predicates in ASP need an additional parameter for the timestep. Furthermore, each action is allowed to execute under the same condition, and makes the same changes to each allowed state, regardless of the language. Table 1 lists all the non-action predicates in this domain.

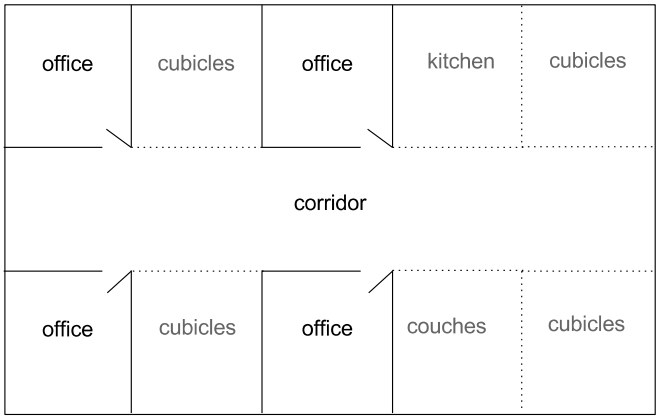


Figure 2: Small office domain.

A predicate is marked as static if its value does not change after initialization, otherwise the value of the predicate depends on the timestep. In ASP, the timestep is modeled using the integer variable `n`, and all dynamic predicates require this parameter. An inertial predicate inherits the value from the previous timestep unless it is changed explicitly. All PDDL predicates are inertial by default. In ASP, for example, the inertial property of `at` is enforced by the following rule:

```
at(R,n) :- at(R,n-1), not -at(R,n).
```

The predicate `canopen` describes the state after a robot moves into a position to sense and open a door. Specifically, the robot has to be facing a closed door before opening the door. The predicate is not inertial. In the PDDL description, `(canopen d)` is explicitly set to false as an effect of every action except for `(approach d)`.

The domain also has a recursive predicate `accessible` which describes the accessibility of rooms. Since one corridor may connect to multiple open rooms, moving between any of them is a single point-to-point navigation action without going through doors. The recursive property can be described as, if `R3` is accessible from both `R1` and `R2`, then `R2` is accessible from `R1`. The complete and formal definition of the predicate `accessible` in ASP is as follows:

ASP	PDDL	Description	Property
`hasdoor(R,D)`	`(hasdoor ?r - room ?d - door)`	Room `R` has door `D`.	static
`connected(R1,R2)`	`(connected ?r1 - room ?r2 - room)`	Room `R1` is connected to room `R2` without a door.	static
`at(R,n)`	`(at ?r - room)`	The robot is in room `R` at timestep `n`.	inertial
`visited(R,n)`	`(visited ?r - room)`	The robot has already visited the room `R` at timestep `n`.	inertial
`open(D,n)`	`(open ?d - door)`	Door `D` is open at timestep `n`.	inertial
`canopen(D,n)`	`(canopen ?d - door)`	The robot is in front of door `D` at timestep `n`.	dynamic
`accessible(R1,R2,n)`	`(accessible ?r1 - room ?r2 - room)`	The robot can navigate from room `R1` to room `R2` with a single navigation action.	recursive

Table 1: Predicates of the office domain.

```
accessible(R1,R2,n) :- connected(R1,R2).
accessible(R1,R2,n) :- open(D,n), hasdoor(R1,D), hasdoor(R2,D).
accessible(R1,R2,n) :- accessible(R2,R1,n).
accessible(R1,R2,n) :- accessible(R1,R3,n), accessible(R2,R3,n).
```

The equivalent definition of `accessible` in PDDL is supported as a derived predicate:

```
(:derived (accessible ?r1 - room ?r2 - room)
  (or (connected ?r1 ?r2)
      (exists (?d - door) (and (open ?d)
                               (hasdoor ?r1 ?d)
                               (hasdoor ?r2 ?d)))
      (accessible ?r2 ?r1)
      (exists (?r3 - room) (and (accessible ?r1 ?r3)
                                (accessible ?r2 ?r3)))))
```

There are three actions in the domain: `go_to`, `approach`, and `open_door`, with the following definitions:

- `go_to`: The action has one parameter: a room `R2`. During execution, the robot follows a path computed by a lower level controller to the

location of `R2`. The precondition is, if the current location is `R1`, the goal location `R2` must be accessible from `R1` with a single navigation action. Once the robot finishes the action, it is no longer in `R1`, and `R2` has been visited. The precondition and effects are expressed as follows in ASP:

```
:- goto(R2,n), at(R1,n-1), not accessible(R1,R2,n-1).
at(R2,n)      :- goto(R2,n).
-at(R1,n)     :- goto(R2,n), at(R1,n-1), R1 != R2.
visited(R2,n) :- goto(R2,n).
```

The equivalent definition in PDDL is as follows:

```
(:action goto
  :parameters (?r2 - room)
  :precondition (exists (?r1 - room)
                        (and (at ?r1)
                        (accessible ?r1 ?r2)))
  :effect (and (at ?r2)
               (forall (?r1 - room)
                       (when (at ?r1)
                             (not (at ?r1))))
               (visited ?r2)
               (forall (?d1 - door)
                       (not (canopen ?d1)))))
```

- `approach`: The action allows the robot to move in front of and face a door `D`, so the robot can sense if the door is open. The robot has to be in a room that has the door `D` before executing `approach D`. The rules are represented in ASP as:

```
:- approach(D,n), at(R1,n-1), not hasdoor(R1,D).
canopen(D,n) :- approach(D,n).
```

The complete PDDL definition of the action is as follows:

```
(:action approach
  :parameters (?d - door)
  :precondition (exists (?r1 - room)
                        (and (at ?r1)
                             (hasdoor ?r1 ?d)))
  :effect (and (canopen ?d)
               (forall (?d1 - door)
                       (when (not (= ?d1 ?d))
                             (not (canopen ?d1))))))
```

- `open_door`: This action specifies that the robot opens a door `D`. The precondition is that the robot is in a pose that allows it to perform the action. The effect is that the door `D` is open. The implementation of this action depends on robot capabilities. The ASP encoding of this action is as follows:

```
:- opendoor(D,n), not canopen(D,n-1).
open(D,n) :- opendoor(D,n).
```

  Similarly, the action is defined in PDDL as:

```
(:action opendoor
  :parameters (?d - door)
  :precondition (canopen ?d)
  :effect (and (open ?d)
               (forall (?d1 - door)
                       (not (canopen ?d1)))))
```

3.2.2 Experiments

The experiments evaluate the performance of two solvers with various configurations of robot navigation problems. Two configurations of domain sizes are used in the experiments. The small domain has 4 offices connected with doors, and 6 directly connected rooms. The large domain has a total of 15 rooms in which 6 are offices that have doors. In each problem, the robot starts in the corridor, and needs to visit a list of randomly selected rooms. The plan length is strongly correlated but not directly proportional to the number of rooms visited, since entering rooms connected via doors takes more steps. In order to capture the underlying distribution, we repeat the trials 50 times for each number, and average the results.

Since International Planning Competition (IPC) in 2014 does not take into account or announce the solving time of each planner, PDDL planners are designed to use all the allowed time (1800 seconds). We therefore select the winner of IPC 2011, FastDownward [16] with the setting LAMA-2011 [31], which supports derived predicates. For the ASP solver, we use the incremental mode of CLINGO-4.5.4 [12]. CLINGO is an Answer Set solving system that integrates CLASP, the winner of the fifth Answer Set Programming Competition in 2015 [5]. Experiments are run on a general-purpose High Throughput Computing (HTC) cluster using only machines with at least 8GB memory.

Figure 3 plots average planning time on the logarithmic scale with standard error. The x-axis shows the number of rooms planned to visit. We

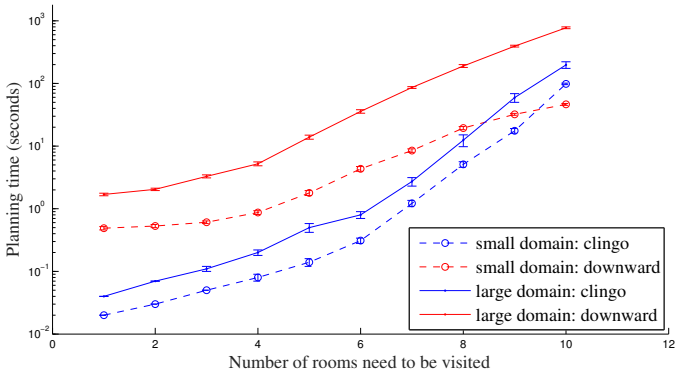


Figure 3: Performance of planners

can observe that when fewer than 8 rooms are visited, the ASP solver is much faster than the PDDL planner, and finds the optimal plan in almost real-time. However, the slope of the red curves is smaller, i.e. the PDDL planner eventually outperforms the ASP solver at large plan length. We can also observe that the ASP solver is less sensitive to the increase in domain size, as the gap between blue curves is smaller than the gap between red curves.

3.2.3 Summary

Based on an empirical comparison between ASP and PDDL solvers, we conclude that ASP-based planning is faster at solving robot navigation problems in large domains, whereas the PDDL planner is better for problems that require long plans. The environment of the BWI robots involves many rooms, doors, objects and people. Any useful encoding of a portion of the CS building exceeds the size of the test domains. Furthermore, task planning is at the most abstract level of their decision making procedures, so plans are more likely to be short. According to the findings of the empirical comparison, we choose ASP for encoding the domain, and CLINGO as the task planner.

4 Multi-robot Planning Algorithm

The second part of the thesis presents a multi-robot task planning algorithm which plans for a team of mobile robots in the same space. In addition to the thesis advisor, Peter Stone, this research is joint work with Shiqi Zhang and Guni Sharon [41].

4.1 Problem Statement

The central planner takes input from each robot of a planning problem that consists of the following information:

- goal state
- current state
- domain description, which includes:
  - definition of predicates and actions, such as the rules introduced in Section 3.
  - static objects and relations.
  - a distribution for the duration of each allowed action.
  - a cost function that assigns a cost to each allowed action. For navigation actions, the cost is often defined by the expected duration.

The output of the central planner is a set of plans in which each plan is a sequence of actions for a robot. The goal of this research is to find the set of plans which minimizes the expected total costs. We make the following assumptions:

- Each robot works on its own task. Tasks are not transferable.
- Robots have real-time communications with the central controller.
- Interactions among robots happen in two cases. First, there is a conflict when two robots are going in the opposite direction in a narrow corridor. Second, robots can take advantage of the `opendoor` action of another robot by following it through. Note that the plan can have non-navigation actions, but interactions only occur in navigation.

4.2 High-level Description of Our Solution

The problem of finding the optimal plans for multiple mobile robots is NP-hard [36]. The robots will not be able to interact in real-time if the symbolic planner has to solve for a formalization which encodes all robots at once. In order to approach the optimal solution in a reasonable amount of time, we first solve a simplified version of the problem: since the central planner knows the planned actions of all robots in the team, when a robot receives a goal and joins the collective, it can optimize the new robot's behavior accordingly. Therefore, the first part of the algorithm focuses on how the knowledge of external actions can be incorporated into the planning process. Section 4.3 shows our approach to adjust the cost function of actions to encode the predicted effects of other robots' actions. An optimal symbolic planner can then compute the optimal plan based on the new cost function. We call this process *conditional planning*, and the optimal plan under the knowledge of the plans of other robots the *conditional optimal plan*. Section 4.4 proposes *iterative conditional planning* (ICP) algorithms that iteratively improves the plan quality with conditional planning.

4.3 Conditional Plan Costs

In this section, we investigate how conflicting navigation actions and sharing door actions affect the expected cost of a plan. The consequence of a head on confrontation of two robots in a narrow corridor is modeled by a large constant collision cost μ. Since temporal uncertainties propagate from previous delays in navigation actions, collisions do not happen deterministically. We derive the expected collision cost $C_{collision}$ for any pair of conflicting actions that navigate the same corridor in opposite directions.

Let T_1^s, T_1^c be the start and completion time of the action being planned, and T_2^s, T_2^c be the start and completion time of the external action. Let their probability density functions (PDF) be $f_{T_1^s}(t)$, $f_{T_1^c}(t)$, $f_{T_2^s}(t)$, and $f_{T_2^c}(t)$. The PDFs are propagated from distributions of previous action durations in the plan (as explained in Section 4.5.1). Since the collision will not happen if one robot enters the corridor after the other robot leaves, the collision cost is defined by the following piecewise function of the four random variables:

$$cost(\text{collision} \mid T_1^s, T_1^c, T_2^s, T_2^c) = \begin{cases} 0 & \text{if } T_1^s > T_2^c \text{ or } T_2^s > T_1^c, \\ \mu & \text{otherwise.} \end{cases} \tag{1}$$

$C_{collision}$ is equal to the expected value of the cost function:

$$
\begin{aligned}
C_{collision} &= \mathbb{E}[cost(\text{collision} \mid T_1^s, T_1^c, T_2^s, T_2^c)] \\
&= \mu \cdot \mathbb{P}(\text{collision} \mid T_1^s, T_1^c, T_2^s, T_2^c) \\
&= \mu \cdot (1 - \mathbb{P}(T_1^s > T_2^c) - \mathbb{P}(T_2^s > T_1^c))
\end{aligned} \tag{2}
$$

Since T_1^s is independent of T_2^c, and T_2^s is independent of T_1^c, it is easy to calculate the joint probabilities by integrating the product of individual PDFs. Therefore, $C_{collision}$ of the action pair is equal to the following expression:

$$
\mu \cdot (1 - \int_0^\infty \int_{t_2}^\infty f_{T_1^s}(t_1) f_{T_2^c}(t_2)\, dt_1 dt_2 - \int_0^\infty \int_{t_1}^\infty f_{T_1^c}(t_1) f_{T_2^s}(t_2)\, dt_2 dt_1) \tag{3}
$$

We then consider the behavior of sharing doors. The action `wait_for_open` allows one robot to wait next to a door which another robot is trying to open. When the other robot completes `open_door`, the waiting robot will move closer to the door and starts going through as soon as the doorway is clear. The action is defined by the following rules in ASP:

```
:- waitforopen(D,n), at(R,n-1), not hasdoor(R,D).
:- waitforopen(D,n), 0{exopendoor(R,D,I)}0.
open(D,n) :- waitforopen(D,n).
facing(D,n) :- waitforopen(D,n).
```

In short, `wait_for_open` is allowed only when there is an external action that opens the door, and the robot is in an area that connects to the door. When the action finishes, the door is open, and the robot is facing the door. The timing of the two actions is again very important. The robot should only plan to wait if the other robot can get a head start. As some doors require holding to remain open, the waiting robot also needs to arrive before the first robot finishes `open_door`. Failure to arrive in the time window will force the robot to open the door by itself. We represent the penalty of such failure by ω, which is often the same as the cost of opening the door again. Suppose T_1^s is the start time of the robot action `wait_for_open`, T_2^s and T_2^c are the start and completion time of the external `open_door` action. The cost function of `wait_for_open` is defined as:

$$
cost(\text{wait} \mid T_1^s, T_2^s, T_2^c) = \begin{cases} \omega & \text{if } T_1^s < T_2^s \text{ or } T_1^s > T_2^c, \\ T_2^c - T_1^s & \text{if } T_2^s < T_1^s < T_2^c. \end{cases} \tag{4}
$$

The expected cost of waiting C_{wait} is calculated as follows:

$$\begin{aligned} C_{wait} &= \mathbb{E}[cost(\text{wait} \,|\, T_1^s, T_2^s, T_2^c)] \\ &= \omega \cdot \mathbb{P}(\text{failure} \,|\, T_1^s, T_2^s, T_2^c) \\ &\quad + \int_0^\infty \int_0^\infty \int_{t_2}^{t_3} (t_3 - t_1) f_{T_1^s}(t_1) f_{T_2^s, T_2^c}(t_2, t_3)\, dt_1 dt_2 dt_3 \end{aligned} \tag{5}$$

where $f_{T_2^s, T_2^c}(t_2, t_3)$ is the joint density function of when the other robot starts and finishes `open_door`. If the cost of door opening is modeled as a constant without uncertainty, T_2^c can be represented by T_2^s to eliminate one variable in Equation 5.

$\mathbb{P}(\text{failure} \,|\, T_1^s, T_2^s, T_2^c)$ is the probability of arriving outside of the valid time window, given by

$$\begin{aligned} \mathbb{P}(\text{failure} \,|\, T_1^s, T_2^s, T_2^c) &= \mathbb{P}(T_1^s < T_2^s) + \mathbb{P}(T_1^s > T_2^c) \\ &= \int_0^\infty \int_0^{t_2} f_{T_1^s}(t_1) f_{T_2^s}(t_2)\, dt_1 dt_2 \\ &\quad + \int_0^\infty \int_{t_3}^\infty f_{T_1^s}(t_1) f_{T_2^c}(t_3)\, dt_1 dt_3 \end{aligned} \tag{6}$$

In practice, the planner can approximate C_{wait} by

$$C_{wait} \approx \mathrm{E}[T_2^c - T_1^s] = \mathbb{E}[T_2^c] - \mathbb{E}[T_1^s] \tag{7}$$

while using a threshold on $\mathbb{P}(\text{failure} \,|\, T_1^s, T_2^s, T_2^c)$ to account for temporal uncertainties.

Algorithm 1 presents a function $C_p(p, A)$ that computes the conditional expected cost of a plan p, given a set A of navigation actions planned by other robots. We can then compute the *conditional optimal plan* by calling an optimal planner to minimize $C_p(p, A)$. Note that the planner can be chosen according to features of the planning problem, as Section 3.2 shows.

Note that the plan might have other types of actions, and the algorithm is generalizable to other kinds of interactions. The collision cost can be used wherever a pair of actions are not allowed to overlap in time due to limited resources. We can also adapt the door sharing calculations to cases in which actions across different robots have to happen in a certain order.

Algorithm 1 Computing conditional plan cost

Input: Plan p, whose cost will be evaluated
Input: Set of external actions A, on which the cost of p is conditioned
Output: C_p: overall expected cost of plan p

```
1: C_p ← 0
2: for each a_1 ∈ p do
3:    if a_1 is not wait_for_open then
4:       C_p ← C_p + cost(a_1)
5:    end if
6:    for each a_2 ∈ A do
7:       if a_1 is wait_for_open(D) and a_2 is open_door(D) then
8:          C_p ← C_p + C_wait(a_1, a_2)
9:       else if a_1 and a_2 are conflicting navigation actions then
10:         C_p ← C_p + C_collision(a_1, a_2)
11:      end if
12:   end for
13: end for
14: return C_p
```

4.4 Iterative Conditional Planning

Our next step is to apply the *conditional planning* method of each robot to approach the global optimum of the system. We present two iterative conditional planning (ICP) algorithms: *Simple ICP* and *Enhanced ICP*.

4.4.1 The Simple ICP Algorithm

Algorithm 2 shows a basic algorithm which iterates through every robot, and computes the optimal plan conditioned on the plans of previous robots. We call this algorithm *Simple Iterative Conditional Planning (S-ICP)*.

Algorithm 2 S-ICP algorithm

Input: N: number of robots
Input: Planning problem for each robot
Output: $p_1, p_2, \cdots, p_N$

```
1: Initialize an empty set of external actions A
2: for each j ∈ {1, 2, ..., N} do
3:    p_j ← the conditional optimal plan under the objective function C_p(p_j, A)
4:    A ← A ∪ {navigation actions in p_j}
5: end for
6: return p_1, p_2, ··· , p_N
```

We can observe that, once a robot selects its plan, it expands the set of external actions for all previous robots. Under the new condition, their chosen plans can be sub-optimal. Thus, the quality of the output depends on the ordering of robots. One way to improve the optimality of S-ICP is to try all orderings, and select the plans with the lowest expected total costs. We call this modified version *Brute-force S-ICP*.

4.4.2 The Enhanced ICP Algorithm

We propose a novel algorithm called *Enhanced Iterative Conditional Planning (E-ICP)* which repeats the *conditional planning* iterations to further reduce total costs of the plans. The algorithm is inspired by simulated annealing, a well-known iterative algorithm for global optimization. The concept of "temperature" in simulated annealing is integrated as the level of importance of other robots, or *negotiation depth*. The central planner initially accepts worse solutions that do not fully adapt to plans of other robots, and moves towards complete collaboration. We implement the negotiation depth as a value α that increases from 0 to 1, where 0 means the plan of each robot is computed independently. At $\alpha = 1$, the plan of each robot is optimized by the full conditional cost (the output of Algorithm 1).

The increase of the importance of other robots is achieved by adjusting their effect on the objective function. In the case of conflicting navigation actions, the algorithm is increasingly concerned about the conflict when the negotiation depth increases. Therefore, we multiply the collision cost in Equation 5 by α. The same reasoning suggests that the collaboration on opening doors should be less favorable in earlier iterations, with the effect of external `open_door` actions completely neglected in the first iteration. This is implemented by elevating the cost of `wait_for_open` towards the cost of opening the door again. The central planner assumes that the collaboration will always fail when $\alpha = 0$, and as α increases to 1, the probability of failure decreases linearly from 1 to the value calculated by Equation 6. Algorithm 3a shows the adjusted cost function which incorporates α in the cost of collision and the cost of collaboration failure.

Algorithm 3b presents the E-ICP algorithm. We introduce an outer loop which iterates from 0 to a parameter Θ. The negotiation depth α starts at 0 and increases by a step size of $\frac{1}{\Theta}$ until it reaches 1 in the final iteration. The inner loop iterates through each robot, computes the *conditional optimal plan*, and maintains the set of navigation actions currently planned by other robots. When $N > 2$, the algorithm has the choice of conditioning the plan on a subset of the team. We introduce an additional parameter M to

represent the size of the subset. Line 6 requires an optimal symbolic task planner to minimize the objective function given by Algorithm 3a.

Algorithm 3a Computing conditional plan cost in E-ICP

Input: Plan p, whose cost will be evaluated
Input: Set of external actions A, on which the cost of p is conditioned
Input: α: negotiation depth
Input: ω : penalty of failure (often the same as the cost of opening the door again)
Output: C_p: overall cost of plan p

1: $C_p \leftarrow 0$
2: **for each** $a_1 \in p$ **do**
3: **if** a_1 is not `wait_for_open` **then**
4: $C_p \leftarrow C_p + cost(a_1)$
5: **end if**
6: **for each** $a_2 \in A$ **do**
7: **if** a_1 is `wait_for_open(D)` and a_2 is `open_door(D)` **then**
8: $C_p \leftarrow C_p + C_{wait}(a_1, a_2) + \boldsymbol{(1-\alpha)\cdot\omega\cdot P_{fail}(a_1, a_2)}$
9: **else if** a_1 and a_2 are conflicting navigation actions **then**
10: $C_p \leftarrow C_p + \boldsymbol{\alpha}\cdot C_{collision}(a_1, a_2)$
11: **end if**
12: **end for**
13: **end for**
14: **return** C_p

Algorithm 3b E-ICP algorithm

Input: N: number of robots
Input: Planning problem for each robot
Input: Θ: number of rounds of "negotiations", $\Theta > 0$
Input: M: number of robots considered in conditional planning, $M \leq N-1$
Output: $p_1, p_2, \cdots, p_N$

1: Initialize an empty set of external actions A
2: **for each** $i \in \{0, 1, \ldots, \Theta\}$ **do**
3: $\alpha = i/\Theta$
4: **for each** $j \in \{1, 2, \ldots, N\}$ **do**
5: $A \leftarrow$ {navigation actions in M other robots}
6: $p_j \leftarrow$ the conditional optimal plan under objective function $C_p(p_j, A, \alpha)$
7: **end for**
8: **end for**
9: **return** $p_1, p_2, \cdots, p_N$

4.4.3 Complexity and Optimality of ICP

The Simple Iterative Conditional Planning algorithm calls the symbolic planner N times. There are $N!$ possible orderings of the N robots. Therefore, the time complexity of Brute-force S-ICP is $O(N! \cdot N)$ with respect to a single operation of conditional planning. Since Enhanced ICP has Θ rounds of conditional planning, its complexity is $O(\Theta \cdot N)$. As the number of robots in the team grows, E-ICP becomes a much more efficient algorithm than Brute-force S-ICP.

Neither Brute-force S-ICP nor E-ICP always outputs the optimal solution. Further, both algorithms can be outperformed by the other. E-ICP often finds better plans if several rounds of negotiations are required to reach the global optimum. In such problems, since S-ICP only plans once for each robot, every ordering generates suboptimal plans. In general, ICP algorithms are suboptimal because robots are "selfish" in conditional planning – each robot receives the best plan for itself under information about the others. E-ICP can get stuck at a local optimum when one robot has a plan that can lower the cost of the system, but has no incentive to switch to it. S-ICP (with best ordering) can sometimes correct this problem by forcing that "selfish" robot to plan last.

4.5 Evaluation

In this section, we evaluate the proposed E-ICP algorithm in the office domain of the BWI robots. To fully instantiate the planning problem stated in Section 4.1, we first assign each navigation action a model of its duration. Quantitative experiments are conducted with a realistic multi-robot simulator GAZEBO [22] and an abstract simulator. We also demonstrate the successful implementation of the algorithm in a trial with real robots.

4.5.1 Navigation Duration

To model the noisy duration of navigation actions in the test domain, we make two additional assumptions:

1. Navigation actions are delayed by independent appearances of humans at a constant rate λ.

2. Each person needs a fixed time δ to pass the blocking zone. In other words, each appearance delays the action by δ.

Following the first assumption, the number of delays per unit time can be modeled by a Poisson distribution with parameter λ.

Proposition: If X and Y are independent Poisson random variables: $X \sim Poisson(\lambda_1)$ and $Y \sim Poisson(\lambda_2)$, then their sum $Z = X + Y$ also follows a Poisson distribution: $Z \sim Poisson(\lambda_1 + \lambda_2)$ [21].

Suppose the navigation action a takes $t_{act}(a)$ units of time at full speed without delays. Following the proposition, we can model the number of delays $N_{delay}(a)$ by a Poisson distribution with parameter $\lambda \cdot t_{act}(a)$. In other words, the action duration $t(a)$ can be modeled by

$$t(a) = t_{act}(a) + \delta \cdot N_{delay}(a) \tag{8}$$

where $N_{delay}(a) \sim Poisson(\lambda \cdot t_{act}(a))$.

Therefore, the model of navigation duration requires two domain parameters λ and δ, and the acting time of each action t_{act}.

Suppose a plan has the sequence of actions $a_1, a_2, \cdots, a_n$. For any action a_k, we can derive the distribution of its start time T_k^s from the parameters of previous actions $a_1, a_2, \cdots, a_{k-1}$. The distribution is required by Equations 3 and 5 to compute the conditional plan cost.

$$T_k^s = \sum_{i=0}^{k-1} t_{act}(a_i) + \delta \cdot \sum_{i=0}^{k-1} N_{delay}(a_i) \tag{9}$$

In general, the PDF of the sum $\sum_{i=0}^{k-1} N_{delay}(a_i)$ is the convolution of each PDF [28]. As $N_{delay}(a) \sim Poisson(\lambda \cdot t_{act}(a))$, the sum $\sum_{i=0}^{k-1} N_{delay}(a_i)$ also follows a Poisson distribution with parameter $\lambda \cdot \sum_{i=0}^{k-1} t_{act}(a_i)$. Let $t'_{act} = \sum_{i=0}^{k-1} t_{act}(a_i)$. Plugging in the probability mass function (PMF) of Poisson distribution, we have

$$\mathbb{P}(\textstyle\sum_{i=0}^{k-1} N_{delay}(a_i) = x) = {(\lambda \cdot t'_{act})^x e^{-(\lambda \cdot t'_{act})}}/{x!} \tag{10}$$

where $\sum_{i=0}^{k-1} N_{delay}(a_i) = {T_k^s - t'_{act}}/{\delta}$. The distribution of T_k^s is defined by:

$$\mathbb{P}(T_k^s = t) = \begin{cases} 0 & \text{if } t \leq t'_{act}, \\ \frac{(\lambda \cdot t'_{act})\hat{}(t - t'_{act}/\delta) \cdot exp(-\lambda \cdot t'_{act})}{(t - t'_{act}/\delta)!} & \text{if } t > t'_{act}. \end{cases} \tag{11}$$

4.5.2 Gazebo Simulation

We use the GAZEBO simulator to compare the performance of E-ICP with 2 rounds of conditional planning ($\Theta = 2$) against the baseline where plans are generated independently. Figure 4 (**left**) shows a map of the testing environment. S_1 and S_2 (green rectangles) are the initial positions of two robots, and G_1 and G_2 (red ellipses) show their goal positions respectively. The two robots are likely to collide without collaboration in planning. In this case, both S-ICP and E-ICP compute the globally optimal plan: one of the robots changes the direction of its path, which is longer in distance but no longer causes conflicts.

Artificial human walkers are introduced in the environment with behavior stated in Section 4.5.1, simulating temporal uncertainties in navigation actions. Figure 4 (**right**) shows a human walker blocking the path of a robot in GAZEBO.

Figure 5 plots the actual plan execution time of each robot in 45 trials. Table 2 reports the average and standard deviation of the results. We can observe that E-ICP reduces both the overall cost and the variance compared to the baseline. When each robot plans independently, the execution time has a wider spread because the consequence of the collision depends on the exact position of the collision, and which robot turns around first. For example, the blue squares on the left happen when $R2$ goes around after the collision, while $R1$ keeps going on the shortest path.

4.5.3 Abstract Simulation

The physics engine and visualization of GAZEBO help verify that E-ICP is a realistic and significant improvement from the baseline. To run a larger scale of experiments, we design an abstract simulator that, instead of simulating navigation and human walkers, samples navigation costs directly from our model in Section 4.5.1. Figure 6 is a map of the simulation domain. The cost of opening each door ω is 12, and the collision cost μ is 40. We use the abstract simulation environment to evaluate three aspects of E-ICP: the effect of negotiation rounds (Θ), the necessity of modeling uncertainties in navigation durations, and the significance of the subset size (M) in a team larger than 2 robots. There is no direct comparison between Brute-foce S-ICP and E-ICP using this simulator, since results from one domain will not be representative of their average performances.

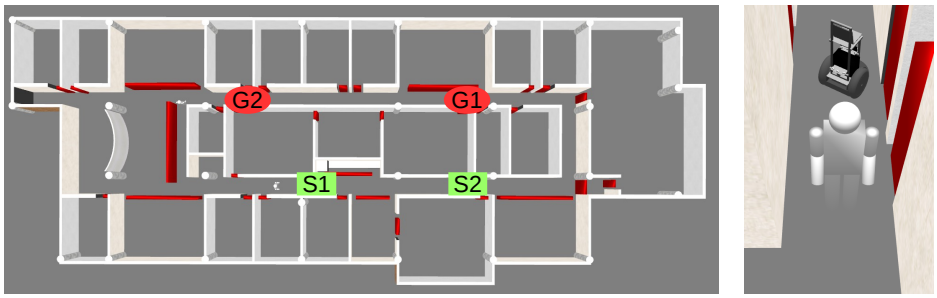


Figure 4: GAZEBO simulation environment (and a picture of a human walker blocking a robot).

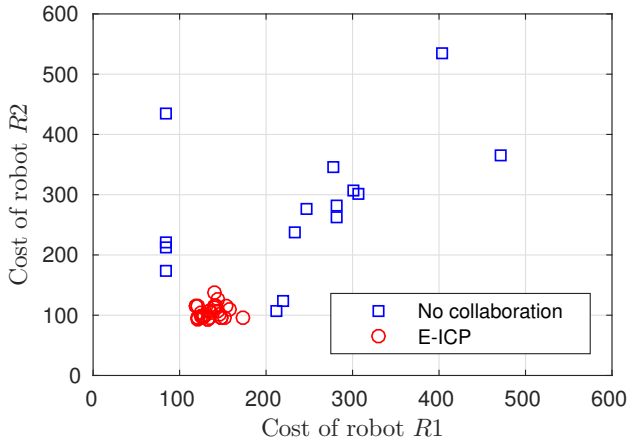


Figure 5: Costs of robots $R1$ and $R2$ in 45 trials collected using GAZEBO simulation environment.

	E-ICP	No collaboration
Robot-1	136.34 (13.18)	238.36 (117.04)
Robot-2	104.85 (10.74)	278.72 (112.55)

Table 2: Average time and standard deviation from GAZEBO simulation experiments.

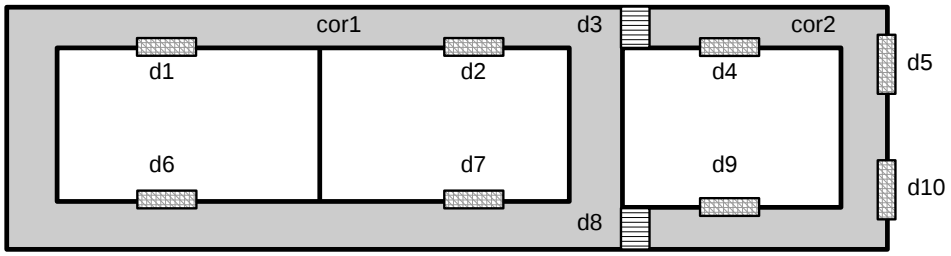


Figure 6: Abstract simulation environment.

Evaluating the effect of Θ We compare the performance of plans generated under $\Theta = 0$, 1, and 2. At $\Theta = 0$, the algorithm has one cycle of planning with $\alpha = 0$, so the plans are equivalent to the individual optimal plans without collaboration. At $\Theta = 1$, the algorithm performs one round of independent planning, and one round of fully conditional planning at $\alpha = 1$, a hybrid of no collaboration and S-ICP. At $\Theta = 2$, we have three rounds of negotiation at $\alpha = 0$, 1 and 2.

Figure 7 reports the average execution costs and the average number of successful door collaborations from 50 trials. In all the trials, $R1$ and $R2$ need to move from $d2$ to $d9$ and from $d7$ to $d4$, respectively. The tasks are assigned so that there is risk of collision (between $d3$ and $d8$), and potential for collaboration on opening doors. The relative starting time is varied in these trials. We can observe that, when one robot starts much earlier than the other, the three configurations have almost the same performance. In cases where the two robots receive the tasks around the same time, $\Theta > 0$ performs significantly better than $\Theta = 0$. Further, the extra round of negotiation at $\Theta = 2$ enables more door collaborations than E-ICP with $\Theta = 1$.

Evaluating the necessity of modeling temporal uncertainties In the next set of experiments, $R1$ plans to go from $d3$ in $corr1$ to $d4$, while $R2$ plans to move from $d7$ to $d5$. There is a risk for $R2$ to give up its shortest path and follow $R1$ through $d3$. If there is any delay in the navigation from $d7$ to $d3$, $R2$ will have to open the door again. Figure 8 shows that, modeling the risk of failure in door collaborations (Equation 6) improves the overall quality of solutions by E-ICP.

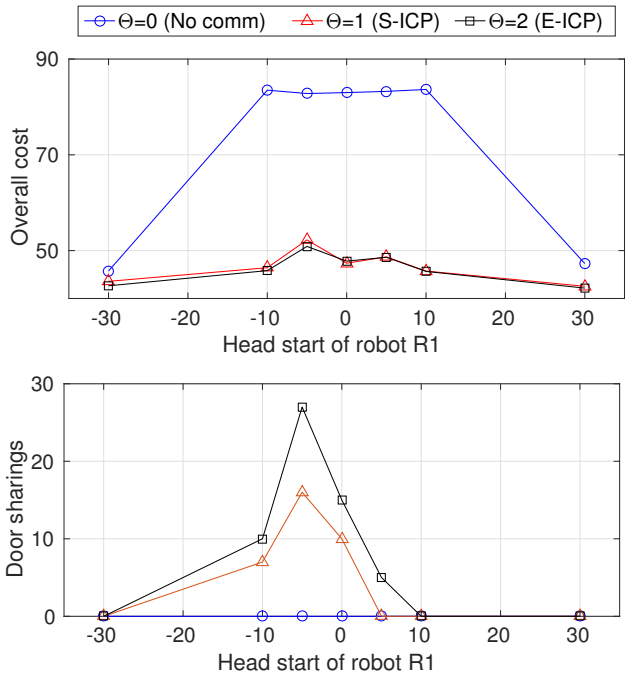


Figure 7: Planning for a two-robot system (evaluating Θ).

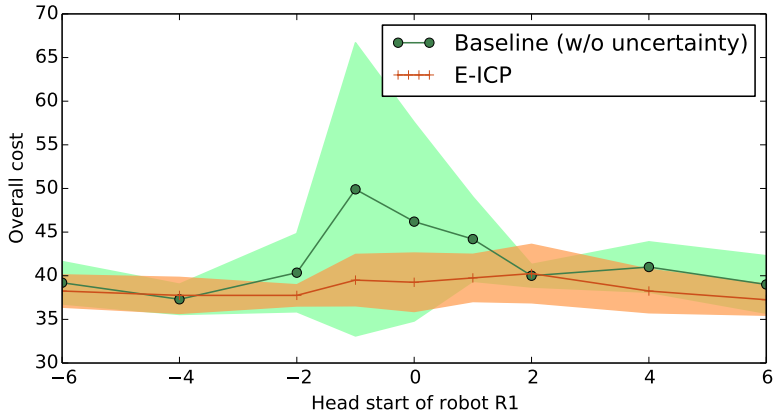


Figure 8: Planning for a two-robot system (evaluating the need of modeling temporal uncertainties).

Evaluating the effect of M Experiments with a team of three robots are conducted in the same simulation environment. The conditional planning step can consider the previous robot ($M = 1$), or all other robots ($M = 2$). We evaluate the effect of M on the performance of E-ICP at $\Theta = 1$ and $\Theta = 2$. In these experiments, $R1$, $R2$ and $R3$ need to move from $d1$ to $d9$, from $d6$ to $d4$, and from $d10$ to $d8$ ($cor1$ side), respectively. Each robot can receive its task at time 0, 15 or 30, generating 19 distinctive initial states. Table 3 shows the mean and standard deviation of the overall costs, based on the 19 states, and 50 trials for each state. The increase of M significantly lowers overall costs (p-value is 0.03 at $\Theta = 1$, and 0.02 at $\Theta = 2$).

4.5.4 Robot Demonstration

Conducting quantitative experiments with multiple real robots can be very costly (in terms of time and possible damage of collisions). Instead, we present the following illustrative trial.

Figure 9 shows the initial ($S1$ and $S2$) and goal positions ($G1$ and $G2$) on the real navigation map of the BWI robots. Without collaboration in task planning, the robots will both have to find someone to open $D1$ and $D2$. The E-ICP algorithm enables the second robot to take advantage of the door opened by the first robot ($D1$). Since the first robot is expected to arrive at $D1$ early, it can go around asking people while the second robot navigates from $S2$ to $D1$. A video of this trial is available at: `https://youtu.be/ADbH3sppLHQ`

	Number of teammates considered in conditional planning	
	$M = 1$	$M = 2$
$\Theta = 1$	207.81 (66.24)	179.28 (9.81)
$\Theta = 2$	205.35 (62.25)	171.03 (9.99)

Table 3: Mean and standard deviation values of the four configurations.

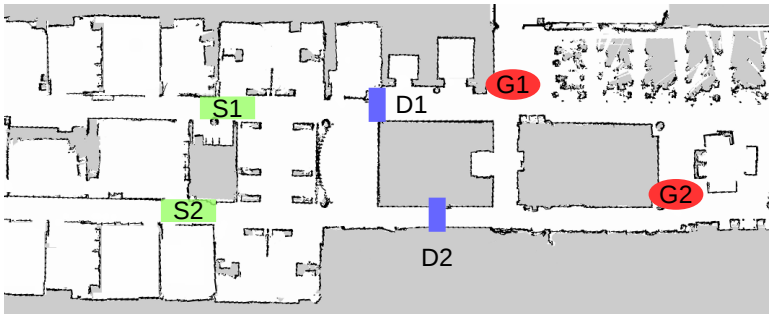


Figure 9: Floor map of the real-world environment.

4.6 Summary

This section proposes the iterative conditional planning algorithm to solve the problem of optimal planning in an environment with multiple mobile robots. The algorithm has two steps: 1) adjusting the objective function in single-robot planning to model influence of other robots, and 2) an iterative algorithm that uses step 1 to approach global optimum. A distribution of navigation durations is derived for the BWI robots, and used to evaluate different configurations of the algorithm.

5 Related Work

Planner competition and comparison: Comparisons of planning algorithms have been mostly provided by the International Planning Competition [35] in the standardized language PDDL [26]. The Answer Set Programming competition evaluates answer set solvers on a variety of domains, in which some of them are planning problems [5]. However, these competitions do not evaluate solvers across different action languages, or analyze domain features that affect the performance of different modeling and solving strategies. Some heuristics of SAT-based planning have been compared to PDDL planners [32]. However, there has not been a direct comparison of ASP-based planning and PDDL-based planning, especially in a realistic robotics domain that requires recursive action effects.

Temporal planning with uncertainties: PDDL has been extended to consider numeric properties including durations of actions in PDDL2.1 [11], and temporal planning problems have been included in International Planning Competitions. Uncontrollable action durations have been expressed as intervals of time, in order to find "strong" plans that are guaranteed to work [6] [27]. Our approach to planning with noisy action durations differ from these methods since 1) we allow domain-dependent probability density functions, and 2) we aim to minimize overall expected costs for a team of robots.

Multi-agent planning: Previous work on multi-agent planning has focused on the representation of concurrent actions in PDDL and other formalisms [2] [1]. When combined with temporal planning, certain extensions of action languages and solvers are capable of planning for multiple agents while considering action durations [3] [7]. However, the direct representation of concurrent action effects significantly increases the complexity of planning [33]. This thesis proposes an algorithm that efficiently plans for a team of robots by modeling concurrent action effects as conditional costs. The algorithm floats above the representation in action languages and the implementation of solvers. Therefore, we can select a fast classical planner based on features of the planning domain, as shown in the first part of this thesis.

Applications of task planning on multiple robots: Symbolic task planning has been applied to a team of robots in many domains [4, 20, 37, 38, 10]. These applications either assume no run-time conflicts [4, 20], or focus on run-time negotiation [37, 38, 10], as opposed to predicting interactions

at planning time.

Multi-robot probabilistic planning: In contrast to symbolic planning, (PO)MDP-based planning has also been explored for multi-robot task planning. Related topics include: planning with concurrent actions [25, 34], planning under temporal uncertainty [15, 40], incorporating temporal logic into navigation task planning [8], and planning for multi-robot systems using a single (PO)MDP [18], multiple (PO)MDPs [42]. The (PO)MDP-based algorithms model non-deterministic action outcomes and aim to optimize long-term expected reward. The symbolic planning paradigm is fundamentally different since the goal is to generate interpretable plans with minimal expected costs, assuming deterministic action outcomes.

6 Conclusion and Future Work

This thesis first investigates the performance of state-of-the-art PDDL and ASP planners in navigation domains. The empirical comparison shows that the PDDL planner is faster at computing long plans, whereas the ASP solver scales better with number of objects. We use the ASP solving system Clingo for planning in the domain of the BWI robots. Based on the existing single-robot planning platform, the second part of the thesis presents an algorithm to efficiently improve overall plan quality for a team of robots.

Possible future work includes generalizing the conditional planning to enable a larger variety of collaborations, and improving the ICP algorithm for better efficiency and optimality. This thesis models navigation durations, and conditional costs between navigation actions. In the real environment, temporal uncertainty also arises in actions that wait for human input, and collaborations may be needed in non-navigation tasks, such as handing off objects to another robot. Further, current evaluation methods are insufficient to compare average performances of Brute-force S-ICP and E-ICP. We may observe that the performance of our solution depends on the accuracy of estimated conditional costs, as well as the optimality of the iterative algorithm. Assuming a cost model, the evaluation of Brute-force S-ICP and E-ICP does not require executing generated plans, since plan quality can be determined by the expected cost. Given more time, I would like to compare the optimality of the two ICP algorithms in a large number of domain configurations, and propose improvements on the algorithm.

7 Acknowledgments

The thesis could not have been possible without the guidance of my advisor Prof. Peter Stone, and collaborators on the two research projects: Dr. Shiqi Zhang, Dr. Guni Sharon, and Piyush Khandelwal. I would also like to thank Dr. Jivko Sinapov for his help on the writing, and Dr. Matteo Leonetti for introducing me to research.

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287, IIS-1637736, IIS-1651089), ONR (21C184-01), and AFOSR (FA9550-14-1-0087). Peter Stone serves on the Board of Directors of, Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

[1] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *IJCAI*, pages 866–873. Citeseer, 1993.

[2] C. Boutilier and R. I. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[3] M. Brenner. Multiagent planning with partially ordered temporal plans. In *IJCAI*, volume 3, pages 1513–1514, 2003.

[4] J. Buehler and M. Pagnucco. A framework for task planning in heterogeneous multi robot systems based on robot capabilities. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[5] F. Calimeri, M. Gebser, M. Maratea, and F. Ricca. Design and results of the fifth answer set programming competition. *Artificial Intelligence*, 231:151–181, 2016.

[6] A. Cimatti, A. Micheli, and M. Roveri. Strong temporal planning with uncontrollable durations: A state-space approach. In *AAAI*, pages 3254–3260, 2015.

[7] M. Crosby and R. P. Petrick. Temporal multiagent planning with concurrent action constraints. In *ICAPS workshop on Distributed and Multi-Agent Planning (DMAP)*, 2014.

[8] J. P. Fentanes, B. Lacerda, T. Krajnik, N. Hawes, and M. Hanheide. Now or later? predicting and maximising success of navigation actions from long-term experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1112–1117. IEEE, 2015.

[9] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[10] I. Filippidis, D. V. Dimarogonas, and K. J. Kyriakopoulos. Decentralized multi-agent control from local ltl specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 6235–6240. IEEE, 2012.

[11] M. Fox and D. Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)*, 20:61–124, 2003.

[12] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

[13] M. Gelfond and V. Lifschitz. Action languages. 1998.

[14] I. Gori, J. Sinapov, P. Khante, P. Stone, and J. K. Aggarwal. Robot-centric activity recognition in the wild. In *Proceedings of the International Conference on Social Robotics (ICSR)*, 2015.

[15] X. Guo and O. Hernández-Lerma. *Continuous-time Markov decision processes*. Springer, 2009.

[16] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligent Research (JAIR)*, 26:191–246, 2006.

[17] J. Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22(3):57, 2001.

[18] P. Khandelwal, S. Barrett, and P. Stone. Leading the way: An efficient multi-robot guidance system. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1625–1633, 2015.

[19] P. Khandelwal, F. Yang, M. Leonetti, V. Lifschitz, and P. Stone. Planning in action language bc while learning action costs for mobile robots. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling*, pages 472–480. Association for the Advancement of Artificial Intelligence (AAAI), 2014.

[20] R. A. Knepper, T. Layton, J. Romanishin, and D. Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 855–862. IEEE, 2013.

[21] O. Knill. Probability and stochastic processes with applications. *Havard Web-Based*, 1994.

[22] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE, 2004.

[23] J. Lee, V. Lifschitz, and F. Yang. Action language bc: Preliminary report. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 983–989. AAAI Press, 2013.

[24] V. Lifschitz. What is answer set programming? In *Proceedings of the 23rd national conference on Artificial intelligence*, pages 1594–1597. AAAI Press, 2008.

[25] Mausam and D. S. Weld. Planning with durative actions in stochastic domains. *J. Artif. Intell. Res.(JAIR)*, 31:33–82, 2008.

[26] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language. 1998.

[27] A. Micheli, M. Do, and D. E. Smith. Compiling away uncertainty in strong temporal planning with uncontrollable durations. In *Proceedings of the twenty-fourth international joint conference on artificial intelligence (IJCAI). AAAI Press*, pages 1631–1637. Citeseer, 2015.

[28] V. Petrov. *Sums of independent random variables*, volume 82. Springer Science & Business Media, 2012.

[29] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: An Open-Source Robot Operating System. In *Open Source Software Workshop*, 2009.

[30] E. A. Quintero, Á. García-Olaya, D. Borrajo, and F. Fernández. Control of autonomous mobile robots with automated planning. *Journal of Physical Agents*, 5(1):3–13, 2011.

[31] S. Richter, M. Westphal, and M. Helmert. Lama 2008 and 2011. In *International Planning Competition*, pages 117–124, 2011.

[32] J. Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[33] J. Rintanen et al. Complexity of concurrent temporal planning. In *ICAPS*, pages 280–287, 2007.

[34] D. E. Smith and D. S. Weld. Temporal planning with mutual exclusion reasoning. In *IJCAI*, volume 99, pages 326–337, 1999.

[35] M. Vallati, L. Chrpa, M. Grzes, T. L. McCluskey, M. Roberts, and S. Sanner. The 2014 international planning competition: Progress and trends. *AI Magazine*, 36(3):90–98, 2015.

[36] G. Wagner and H. Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1 – 24, 2015.

[37] K. W. Wong and H. Kress-Gazit. Let's talk: Autonomous conflict resolution for robots carrying out individual high-level tasks in a shared workspace. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 339–345. IEEE, 2015.

[38] K. W. Wong and H. Kress-Gazit. Need-based coordination for decentralized high-level robot control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[39] F. Yang, P. Khandelwal, M. Leonetti, and P. Stone. Planning in answer set programming while learning action costs for mobile robots. In *2014 AAAI Spring Symposium Series*, 2014.

[40] H. L. Younes and R. G. Simmons. Solving generalized semi-markov decision processes using continuous phase-type distributions. In *The AAAI Conference on Artificial Intelligence*, 2004.

[41] S. Zhang, Y. Jiang, G. Sharon, and P. Stone. Multirobot symbolic planning under temporal uncertainty. 2016.

[42] S. Zhang, M. Sridharan, and C. Washington. Active visual planning for mobile robot teams using hierarchical POMDPs. *IEEE Transactions on Robotics*, 29(4):975–985, 2013.